# sheets Programmeren 2 — Java

Recursie, de muis en graphics

Walter Kosters

http://www.liacs.nl/home/kosters/java/

Java — **intro**

Voorkennis: dat wat bij het college Programmeren 1 over Java behandeld is; zie ook de sheets van dit college (via de website). Voor verdere informatie raadplege men bijvoorbeeld het boek Java voor studenten van D. Bell en M. Parr, Prentice Hall, 2002.

Dit keer behandelen we:

- Recursie: functies kunnen zichzelf aanroepen; denk ook aan het Droste-effect.

- De muis: hoe handel je in een grafische omgeving "muis-events" af?

- Graphics: meer grafische mogelijkheden.

En bij voorkeur alle drie tegelijk!
Een en ander sluit aan op het gastcollege van 13 november 2002.

Java — **recursie**

De meeste moderne programmeertalen kennen het begrip *recursie*: een functie kan zichzelf aanroepen.

Een eerste voorbeeld:

```
public int som (int n) {
  int res = 0;
  if ( n == 1 )
    res = 1;
  else
    res = n + som (n-1);  // recursie!
  return res;
}  // som
```

Deze functie berekent de som van de getallen `1`, `2`, `...`, `n`; bijvoorbeeld voor `n` gelijk aan `5`: `1+2+3+4+5 = 15`. De functie zegt eigenlijk: als `n` gelijk aan `1` is, is het antwoord natuurlijk `1`; anders tellen we `n` op bij de som van de getallen `1`, `2`, `...`, `n-1`.

## Java — **recursie: opmerkingen**

Wat valt op bij recursie?

Er zijn altijd één of meer basisgevallen, waarbij *geen* recursie nodig is. Anders kan het oneindig lang doorgaan: denk maar aan twee tegenover elkaar hangende spiegels, of aan het Droste-effect. In ons voorbeeld is deze basissituatie het geval `n` gelijk aan `1`.

Recursie kan heel vaak vermeden worden. In ons voorbeeld: een simpele for-loop

```
for (int i = 1; i <= n; i++ )
  res = res + i;
```

levert ook het gewenste resultaat.

Probeer niet in detail te volgen hoe de computer de recursie afwikkelt. Denk eerder als volgt: iedere (recursieve) functie-aanroep doet wat hij belooft te doen. Zo geeft in ons voorbeeld `som (n-1)` de som van de getallen `1`, `2`, . . . , `n-1`.

## Java — **de muis**

Hoe reageer je in Java op "muis-events"? Dat gaat schematisch als volgt:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Muis extends Applet
       implements MouseListener,
                  MouseMotionListener {

  public void init ( ) {
    addMouseListener (this);
    addMouseMotionListener (this);
  }  // init

  // en nu reageren op de verschillende events,
  // zoals mouseClicked en mouseDragged

}  // Muis
```

## Java — **de muis: event-handlers**

Het gaat om de volgende "event-handlers":

```
// voor de MouseListener:
public void mouseClicked (MouseEvent event) {
  ... }  // muis geklikt
public void mouseReleased (MouseEvent event) {
  ... }  // muis losgelaten na verslepen
public void mousePressed (MouseEvent event) {
  ... }  // muisknop ingedrukt
public void mouseEntered (MouseEvent event) {
  ... }  // muis komt applet binnen
public void mouseExited (MouseEvent event) {
  ... }  // muis verlaat applet

// en voor de MouseMotionListener:
public void mouseDragged (MouseEvent event) {
  ... }  // muis wordt gesleept
public void mouseMoved (MouseEvent event) {
  ... }  // muis beweegt
```

Ze moeten in principe allemaal aanwezig zijn — eventueel leeg (als je ze niet gebruikt).

## Java — **slepen met de muis**

Een voorbeeldje:

```
public void mouseDragged (MouseEvent event) {
  Graphics g = getGraphics ( );
  g.drawLine (100,100,
      event.getX ( ),event.getY ( ));
}  // mouseDragged
```

Dit tekent tijdens het slepen ("draggen") van de muis, met ingedrukte knop dus, steeds een rechte lijn van de huidige muispositie (opgehaald met `event.getX ( )` en `event.getY ( )`) naar het vaste punt `(100,100)`.

Een functie als `mouseClicked` kan ook heel goed, via `repaint ( )`, de functie `paint ( )` aanroepen. Denk er wel aan dat dan het hele window hertekend wordt — voor slepen levert dat een heel ander effect op!

## Java — **kleurenwaaier**

En zo sleep je een kleurenwaaier achter de muis aan:

```
public void mouseDragged (MouseEvent event) {
  Graphics g = getGraphics ( );
  teller++;
  switch ( teller % 5 ) {
    case 0: g.setColor (Color.red); break;
    case 1: g.setColor (Color.green); break;
    case 2: g.setColor (Color.black); break;
    case 3: g.setColor (Color.blue); break;
    case 4: g.setColor (Color.yellow); break;
  }  // switch
  g.drawLine (100,100,
      event.getX ( ),event.getY ( ));
}  // mouseDragged
```

Hierbij is `teller` een globale variabele.

## Java — **graphics**

In Java zijn talloze mogelijkheden om mooie graphics te genereren, denk aan het gastcollege van 13 november 2002. Je kunt eventueel `swing` gebruiken, een verbeterde versie van `awt` (de Abstract Window Toolkit). Zo worden `swing`-windows wel ververst als er tijdelijk een ander window overheen geparkeerd is geweest, kun je plaatjes in de knoppen krijgen, enzovoorts. Wij beperken ons tot `awt`.

Ons doel is een applet te maken waarin fraaie grafische sneeuwkristallen staan — die verschoven kunnen worden.

De functie

```
public void paint (Graphics g) {
  g.setColor(Color.white);
  g.fillRect(0,0,400,400);
}  // paint
```

levert al vast een mooie witte achtergrond op.

## Java — **recursief tekenen**

Onze eerste recursieve functie die iets tekent, maakt steeds kleiner wordende in elkaar gelegen rechthoeken:

```
public void tekenrec (int x, int y,
    int breedte, int hoogte) {
  Graphics g = getGraphics ( );
  g.setColor (Color.red);
  g.drawRect (x,y,breedte,hoogte);
  if ( hoogte > 6 && breedte > 6 ) {
    tekenrec (x + (int) (0.17 * breedte),
              y + (int) (0.17 * hoogte),
              (int) (0.67 * breedte),
              (int) (0.67 * hoogte));
  }  // if
}  // tekenrec
```

## Java — **recursief tekenen: details**

Er wordt een rode rechthoek getekend met hoekpunt linksboven op positie `(x,y)`, en met breedte `breedte` en hoogte `hoogte`.
Als de hoogte en de breedte nog wat voorstellen, tekenen we een factor $2/3 \approx 0.67$ verkleinde rechthoek, een stukje verschoven binnen de oorspronkelijke. Door de recursie gaat dit steeds maar door . . .

Er is dus geen basisgeval — of preciezer: in het basisgeval doen we niks.

Deze functie kan overigens ook eenvoudig met een for-loop geprogrammeerd worden.

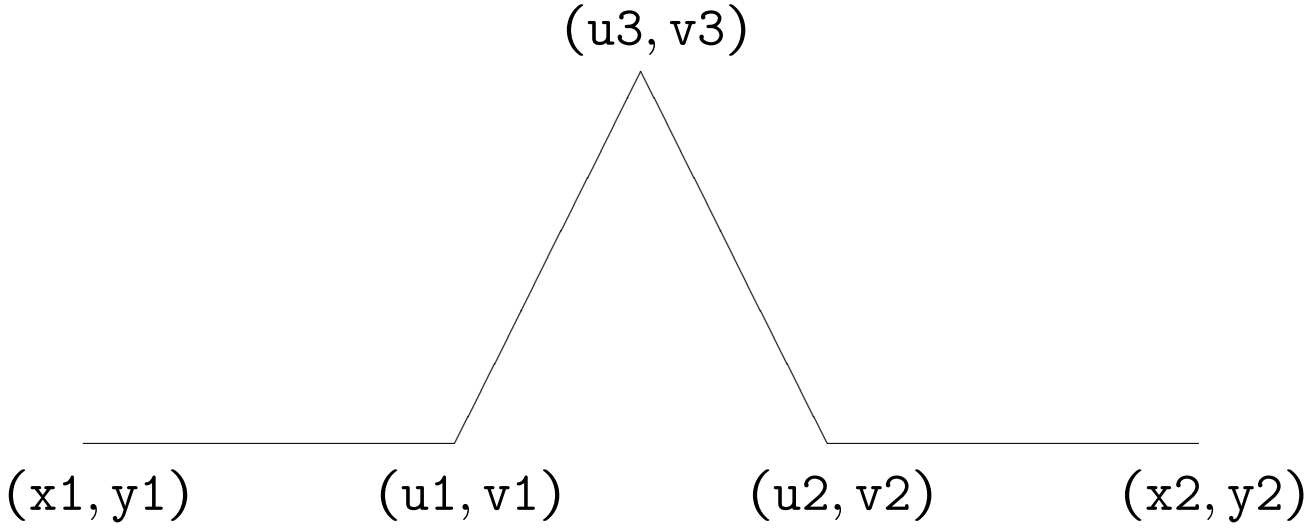
## Java — **Koch-kromme**

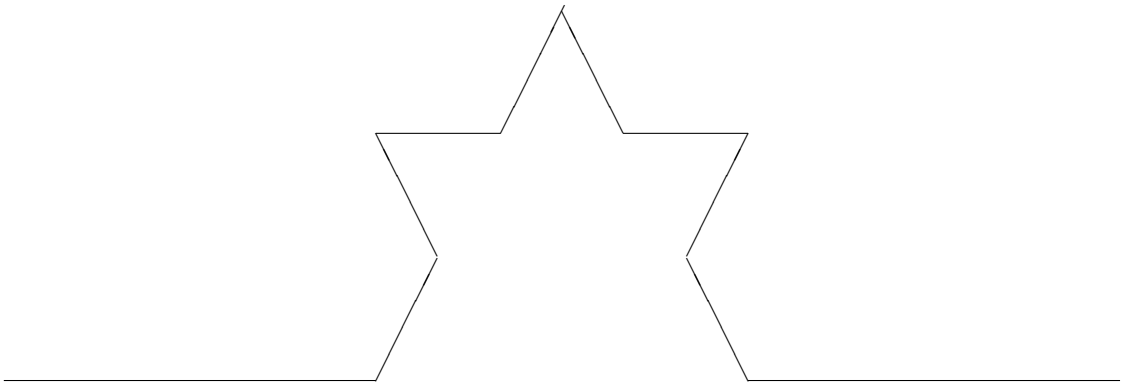
En nu een echt ingewikkelde recursieve functie, die de "Koch-kromme" oplevert (let maar niet op de berekening van `u1`, `v1`, `u2`, `v2`, `u3` en `v3`):

```
public void koch (Graphics g, int nivo,
    int x1, int y1, int x2, int y2) {
  int u3 = (int) (0.289*(y2-y1)) + (x1+x2)/2;
  int v3 = (int) (0.289*(x1-x2)) + (y1+y2)/2;
  int u1 = (2*x1+x2)/3, v1 = (2*y1+y2)/3;
  int u2 = (x1+2*x2)/3, v2 = (y1+2*y2)/3;
  if ( nivo == 1 )  // basisgeval
    g.drawLine (x1,y1,x2,y2);
  else {
    g.drawLine (x1,y1,u1,v1);
    koch (g,nivo-1,u1,v1,u3,v3);
    koch (g,nivo-1,u3,v3,u2,v2);
    g.drawLine (u2,v2,x2,y2);
  }  // else
}  // koch
```

## Java — **Koch-kromme: plaatjes**

Het volgende plaatjes ontstaan bij aanroep van de functie `koch` met `nivo` gelijk aan `2` en `3` (coördinaten zijn ter illustratie toegevoegd):



## Java — **Koch-kromme: rest**

En het “gehele” programma:

```
public int xCo = 20;
public int yCo = 250;
public int teller = 1;
public void paint (Graphics g) {
  g.setColor(Color.white);
  g.fillRect(0,0,400,400);
  g.setColor(Color.green);
  koch (g,teller,xCo,yCo,xCo+350,yCo);
}  // paint
public void mouseClicked (MouseEvent event) {
  xCo = event.getX ( );
  yCo = event.getY ( );
  teller++;
  repaint ( );
}  // mouseClicked
public void koch (Graphics g, int nivo,
    int x1, int y1, int x2, int y2) {
  // ... zie terug ...
}  // koch
```

## Java — **samengevat**

We hebben gezien hoe je de muis kunt gebruiken in een grafische omgeving, en hoe recursie werkt.

Je kunt —zoals in het `koch`-voorbeeld— ervoor kiezen om met `paint ( )` te werken (eigenlijk door middel van `repaint ( )`), waarbij steeds het hele venster opnieuw getekend wordt, of met `getGraphics ( )` steeds bijtekenen in het huidige venster — zoals bij de kleurenwaaier.

De opgave is nu een applet te schrijven die op muis-events reageert, die een interessante grafische ervaring oplevert, en die bij voorkeur een recursieve functie bevat. Er valt altijd wel iets te verschuiven . . .

Java — **tja**

En voor de liefhebbers:

```
public void sneeuw (Graphics g, int x, int y,
    int b, int h, int richting) {
  int ob = b;
  int oh = h;
  g.drawRect (x,y,b,h);
  b = b / 2;
  h = h / 2;
  if ( b > 5 && h > 5 ) {
    if ( richting != 3 )
      sneeuw (x-b,y-h,b,h,1);
    if ( richting != 1 )
      sneeuw (x+ob,y+oh,b,h,3);
    if ( richting != 4 )
      sneeuw (x+ob,y-h,b,h,2);
    if ( richting != 2 )
      sneeuw (x-b,y+oh,b,h,4);
  } // if
} // sneeuw
```